

Nro. ord.	Apellido y nombre	L.U.	#hojas
9	Brandwein Eric	3491/16	5 + 1

ORGANIZACIÓN DEL COMPUTADOR I - Parcial
Primer Cuatrimestre 2017

Ej. 1	Ej. 2	Ej. 3	Ej. 4	Nota
B	B	B	B	A

Corrector:

Jiepo

Aclaraciones

- Anote apellido, nombre, LU y numere *todas* las hojas entregadas
- Cada ejercicio se califica con Bien, Regular o Mal. La división de los ejercicios en incisos es meramente orientativa. Los ejercicios se califican globalmente.
- El parcial NO es a libro abierto, pero puede tener los apuntes provistos por la cátedra y cinco hojas A4 (10 carillas) con apuntes propios.
- Importante:** Justifique sus respuestas. Las soluciones a ejercicios de la práctica que se utilicen deben ser incluidas en el examen.
- El parcial se aprueba con dos ejercicios Bien y al menos uno Regular.

Ejercicio 1 Se desea diseñar una máquina inspirada en el famoso procesador MIPS R3000. La misma tendrá arquitectura Von Neumann, con tamaño de palabra 32 bits, direccionamiento a byte y 32 registros (R_0 a R_{31}).

El set de instrucciones será el siguiente:

Instrucción	Ejecución	Comentarios
add R_i, R_j, R_k	$R_i = R_j + R_k$	suma, 3 operandos
sub R_i, R_j, R_k	$R_i = R_j - R_k$	resta, 3 operandos
addi R_i, R_j, cte	$R_i = R_j + cte$	suma, 2reg. y constante
addu R_i, R_j, R_k	$R_i = R_j + R_k$	suma sin signo, 3 operandos
subi R_i, R_j, R_k	$R_i = R_j - R_k$	resta sin signo, 3 operandos
addi R_i, R_j, cte	$R_i = R_j + cte$	suma sin signo, 2reg y constante
and R_i, R_j, R_k	$R_i = R_j \& R_k$	and lógico, 3 operandos
or R_i, R_j, R_k	$R_i = R_j R_k$	or lógico, 3 operandos
and R_i, R_j, cte	$R_i = R_j \& cte$	AND registro, constante
or R_i, R_j, cte	$R_i = R_j cte$	OR registro, constante
sll R_i, R_j, cte	$R_i = R_j \ll cte$	Shift izquierda con constante
srl R_i, R_j, cte	$R_i = R_j \gg cte$	Shift derecha con constante
lw $R_i, (cte)R_j$	$R_i = \text{Memory}[R_j + cte]$	Memoria a registro
sw $R_i, (cte)R_j$	$\text{Memory}[R_j + cte] = R_i$	Registro a memoria
lui R_i, cte	$R_i = cte \ll 16$	Carga constante en upper 16bits
beq R_i, R_j, cte	if ($R_i == R_j$) PC = PC + 4 + cte	Salto relativo si son iguales
bne R_i, R_j, cte	if ($R_i != R_j$) PC = PC + 4 + cte	Salto relativo si NO son iguales
slt R_i, R_j, R_k	if ($R_j < R_k$) $R_i = 1$; else $R_i = 0$	Setea si es menor; complemento a 2
slti R_i, R_j, cte	if ($R_j < cte$) $R_i = 1$; else $R_i = 0$	Setea si es menor que constante; complemento a 2
sltu R_i, R_j, R_k	if ($R_j < R_k$) $R_i = 1$; else $R_i = 0$	Setea si es menor; número natural
sltiu R_i, R_j, cte	if ($R_j < cte$) $R_i = 1$; else $R_i = 0$	Setea si es menor que constante; número natural
j cte	PC = cte	Salto
j R_{31}	PC = R_{31}	Salto a posición registro
jal cte	$R_{31} = PC + 4$; PC = cte	Para rutinas

- ¿Cómo se carga una constante de 32 bits en un registro? Mostrar un ejemplo.
- Proponer un Formato de Instrucción de longitud fija de 32 bits. Indicar el tamaño máximo de las constantes involucradas en cada operación.
- Según el formato de instrucción definido, indicar:
 - El tamaño máximo de la memoria
 - La cantidad de instrucciones sin operandos que podrían agregarse a este formato de instrucción.

Ejercicio 2

Hemos sido contratados por una empresa fabricante de lavarropas llamada DEEPWASH. La empresa ha decidido incorporar DeepLearning al lavado de ropa mediante el sensado de suciedad y el procesamiento para la optimización del uso del jabón y velocidad de rotación del tambor. Para esto se utilizará la máquina diseñada en el ejercicio anterior.

- a. Realice el diagrama parcial del *datapath* de una microarquitectura para las instrucciones Load y Add de DEEPWASH. Recuerde indicar el tamaño de cada registro, de los buses internos y externos, las señales de cada componente y justificar la utilización de cada componente escogido y cada decisión tomada. Para realizar el *datapath* puede utilizar los siguientes componentes:

- una ALU que realiza la operación ADD.
- un incrementador que suma 1.
- un controlador de memoria.

Ejercicio 3 El diseño de la DEEPWASH ha sido un éxito y ya se cuenta con un prototipo. Se requiere programar algunos tests. Para ello, primero será necesario definir el mapeo de los sensores y motores en las últimas 16 direcciones de la memoria, reservadas para ello.

- a. Realizar el mapeo para los siguientes dispositivos de E/S: Un sensor de suciedad que usará un registro llamado MUGRE_STAT, un motor con su registro MOTLAVADO_CTRL, un encoder de peso de la ropa con su registro PUN_STAT, un reloj (en segundos) leyendo el registro CLOCK y un dispositivo que recibe la cantidad de suciedad y el peso de la ropa (en el registro OPT_IN) y devuelve el tiempo requerido para una limpieza óptima (en el registro OPT_OUT).
- b. Realizar un diagrama de interconexión del sistema indicando qué tipo de buses se utiliza, la dirección de los buses, las líneas implicadas y la ubicación de los registros de E/S.
- c. Indicar el tamaño de los registros, formato de entrada y salida en cada uno y rango de valores para cada una de las funcionalidades.
- d. Realizar un programa que primero chequee el peso y suciedad de la ropa y arranque el motor durante el tiempo indicado por el dispositivo optimizador. Escribir un Pseudocódigo y programarlo con el Set de Instrucciones de la DEEPWASH.
- e. Se introduce una funcionalidad en DEEPWASH para detectar desbalanceo del tambor. El sensor envía una interrupción al desbalancearse que debe frenar inmediatamente el motor. Indicar en el diagrama de interconexión como interconectar este nuevo dispositivo. Agregar a la arquitectura de DEEPWASH la posibilidad de atender una única interrupción detallando cómo se modifica la arquitectura y el set de instrucciones. Modificar el código anterior para que incorpore esta funcionalidad.

Ejercicio 4 Dado el siguiente código escrito en arquitectura ORGA1 y el vuelco de memoria que se encuentra debajo. Suponiendo que el programa se encuentra cargado a partir de la posición 0x6140. Todos los flags y registros comienzan en 0, excepto el Program Counter cuyo valor inicial es 0x6143 y el Stack Pointer que empieza en 0xFFEE.

```
dato1:  DW 0x1C18
dato2:  DW 0x4532
cont:   DW 0x4533
inicio: MOV R1,dato1
        MOV R2,[dato2]
        ADD R2,[R1]
resto:  SUB [R2],[[cont]]
comparo: CMP R1,R3
        JE fin
inc:    ADD R3,1
        JMP comparo
fin:    MOV [R2],FFFF
        RET
```

	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
4530	F303	ABCD	21E3	6530	8000	1840	95B3	C925	A639	ECBD	FAF0	ABCD	0785	A631	F17B	4925

Se pide:

- a. ¿Cuál es la dirección de memoria denotada por cada etiqueta?
- b. Realizar el seguimiento del programa indicando el estado de los registros, flags y valores del stack en cada paso.

a. Hay más de una forma de hacerlo. Una de ellas es cargar con lui los upper 16 bits, luego hacer un shift con srl, y luego cargar de nuevo con lui los bits restantes.

Por ejemplo, si queremos cargar en el R0 la constante 0x1111222, deberíamos hacer algo así:

lui R0, 0x2222 → ojo con lui. Sobre escribe los 16 bits ^{menos significativos}
 srl R0, R0, 0x10
 lui R0, 0x1111 → hay que sumar el 0x2222 después de cargar el 0x1111

b. Veamos cuánto debería ocuparse cada parte de una instrucción:

Registros: 5 bits c/u ✓

El resto se dedicará instrucción a instrucción. Si cada cuadrado de la cuadrícula representa 1 bit, el espacio se repartiría así:

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
add	C. de "Instrucción"															Ri					Rj					Rk						
sub																Ri					Rj					Rk						
addu																"					"					"						
subi																"					"					"						
addi	Instr.															Ri					Rj					cte.						
andi	código de Instrucción															Ri					Rj					Rk						
ori	código de "Instrucción"															"					"					"						
andi	"Instr."															Ri					Rj					cte.						
ori	"Instr."															Ri					Rj					cte.						
sll	C. de Instrucción															Ri					Rj					cte.						
srl	"															Ri					Rj					cte.						
lw	"Instr."															Ri					cte.					Rj						
sw	"															Ri					cte.					Rj						
lui	C. de Instrucción															Ri					cte.					cte.						
beq	"Instr."															Ri					Rj					cte.						
bne	"															Ri					Rj					cte.						
slt	código de Instrucción															Ri					Rj					Rk						
slti	"Instr."															Ri					cte.					cte.						
sltiu	código de Instrucción															Ri					Rj					Rk						
slti	"Instr."															Ri					cte.					cte.						
j	código de Instrucción															cte.					cte.					cte.						
jR31	código de Instrucción															cte.					cte.					cte.						
jal	código de Instrucción															cte.					cte.					cte.						

El tamaño máximo de las constantes es 16 bits en la mayoría de los casos, excepto sll y srl, en los que es 5 bits.

El código de instrucción de cada una podría ser algo así:

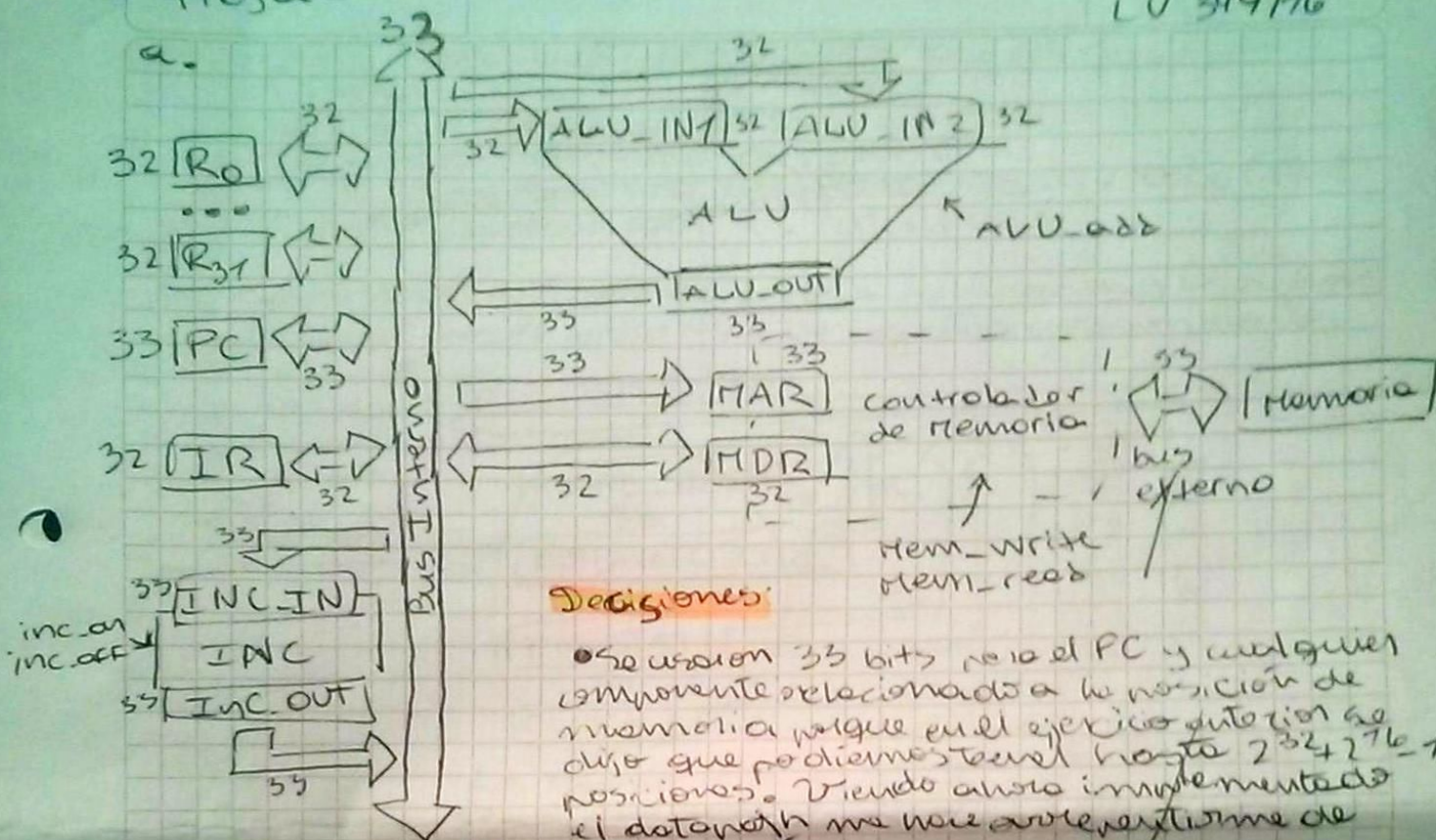
(sigue)

Hoja 2

Ejercicio 2

Brannan Eric
LU 349176

a.

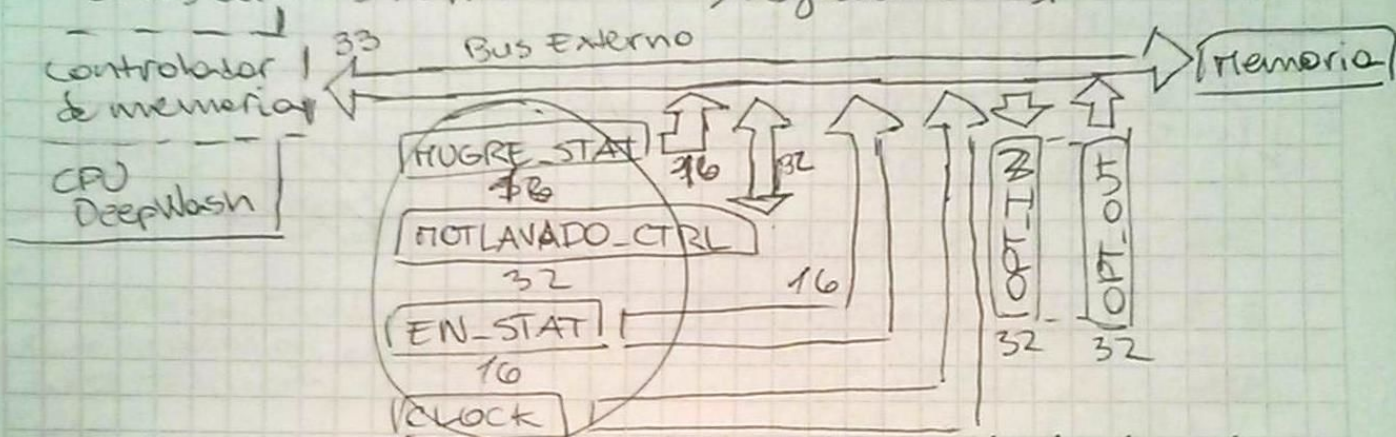


Decisiones:

- Se usaron 33 bits para el PC y cualquier componente relacionado a la posición de memoria porque en el ejercicio anterior se dijo que podríamos tener hasta $2^{32} + 2^{16} - 1$ posiciones. Viendo ahora implementado el datapath me hace aver si me da de ese decisión.
- El INC incrementa de a 1, pero el enunciado, pero como se usó exclusivamente para incrementar el PC, y el mismo se incrementa siempre de a 4, hubiese usado un incrementador de a 4 si hubiese podido.
- En los casos en los que un registro es de 33 bits pero se le pasa un valor de 32 bits, se asume que el bit faltante es el más significativo, y se asume que vale 0.
- La memoria, cuando se quiere leer, recibe una posición de 33 bits, y "devuelve" un valor de un byte. El controlador de memoria se encarga de juntar los bytes en una palabra de 32 bits, y hace lo inverso en el caso de escribir.

- a. MUGRE_STAT \rightarrow ~~0x10000FFF~~ 0x10000FFF9
 MOTLAVADO_CTRL \rightarrow 0x10000FFE
 EN_STAT \rightarrow ~~0x10000FFF~~ 0x10000FFFD
 CLOCK \rightarrow ~~0x10000FFF~~ 0x10000FFFC
 OPT_IN \rightarrow ~~0x10000FFF~~ 0x10000FFFB
 OPT_OUT \rightarrow ~~0x10000FFF~~ 0x10000FFFA

- b. El controlador de memoria debería redireccionar las reads a los últimos 16 posiciones a los registros correspondientes.



Los registros están en cada uno de los dispositivos.

- c. Los registros EN_STAT y MUGRE_STAT son positivos de 16 bits, para que el OPT_IN reciba en los 16 bits más significativos el valor de la ciudad, y en los menos significativos el valor del peso.

MOTLAVADO_CTRL, OPT_OUT y CLOCK son positivos y cero ya que ninguno necesita ser negativo.

MOTLAVADO_CTRL controla la velocidad del motor, 0 queriendo decir "quieto".

- d. Pseudocódigo:

```

Mover MUGRE_STAT y EN_STAT a un solo registro
Mover ese registro a OPT_IN
R2 := CLOCK + OPT_OUT
MOTLAVADO_CTRL := "Prendido"
while (CLOCK < R2):
    no hacer nada
MOTLAVADO_CTRL := "Apagado"
  
```

asumimos que: FFFFFFFF

- "Prendido" = 0x ~~10000000~~, "Apagado" = 0x00000000
- El controlador de memoria devuelve MUGRE_STAT y EN_STAT completados con ceros a la izquierda.
- La suma del 1 y el 5 es 4 en signo.

(signo)

Programa: # : comentario

```
# R0 := 0xFFFFFFFF
# MUGRE-STAT := 0x1000FFFF
# EN-STAT := 0x1000FFFD
# OPT-IN := 0x10000FFF13
# OPT-OUT := 0x10000FFF1A
# CLOCK := 0x10000FFEC
```

```
[ lui R0, 0xFFFF
  srl R0, R0, 0x10
  lui R0, 0xFFFF
  # R1 := MUGRE-STAT | EN-STAT
  lw R1, (0xFFFA) R0
  srl R1, R1, 0x10
  lw R2, (0xFFFE) R0
  addu R1, R1, R2
```

```
sw R1, (0xFFFC) R0
```

```
# R3 := CLOCK + OPT-OUT
```

```
lw R3, (0xFFFD) R0
lw R4, (0xFFFA) R0
addu R3, R3, R4
```

```
# NOT LAVADO-CTRL := 0xFFFF FFFF + 0xFFFF
```

```
sw R0, (0xFFFF) R0
```

```
# R10 := CLOCK, R11 := CLOCK < R3, R30 = 1
```

```
lui R30, 0x0001
srl R0, R0, 0x10
lui R30, 0x0000
```

```
loop: lw R10, (0xFFFD) R0
      sh R11, R10, R3
      beq R11, R30, 0xFFFF
      # -12 en complemento a 2 es 0xFFFF
```

```
# R29 := 0
```

```
lui R29, 0x0000
srl R29, R29, 0x10
lui R29, 0x0000
```

```
sw R29, (0xFFFF) R0
```

```
# FIN.
```


Hoje 4

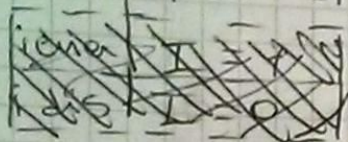
Ejercicio 3
(cont.)

Brandwein Fritz
LU 349/16

El sensor debería conectarse por una conexión separada a la CPU, no por el bus externo, ya que, el ser de máxima prioridad, se necesita la mayor cantidad de protocolo posible para la comunicación con él.

Se podría agregar a la arquitectura un flag I , que indique si se puede interrumpir, y las siguientes instrucciones:

$iret$ | retorna el control de la ejecución al lugar en donde se encontraba anteriormente.
 $PC = PC_i, R_0 \dots R_{31} = R_{0i} \dots R_{31i}, I = 1$



Se deberían también agregar los registros $R_{0i}, R_{1i}, \dots, R_{31i}$, que se usarían para guardar los registros $R_{0..31}$ antes de la interrupción, y PC_i , que guardaría el PC anterior.

Cuando ocurre una interrupción, la arquitectura realiza estos pasos, después de que termine el comando actual:

```
if (I = 1) { I := 0
  PC_i := PC
  PC := 0x00000000
  R_0i ... R_31i := R_0 ... R_31
}
```

}

Código Nuevo:

En la posición $0x00000000$, antes del programa anterior:

$\#R_0 := 0x00000000$

$lui R_0, 0x0000$

$srl R_0, R_0, 0x10$

$lui R_0, 0x0000$

$\#R_1 := 0x00000000$

$lui R_1, 0xFFFF$

$srl R_1, R_1, 0x10$

$lui R_1, 0xFFFF$

(sigue)

NOTLANDO_CTRL := 0xFFFF FFFF + 0xFFFF

SW R0, (0xFFFF) R1

iret

[...]

(programa exterior) <

#FIN

Hoja 5

Ejercicio 4

Erasmus Erit
LU 349/16

a. dato1: 0x6140
dato2: 0x6141
cont: 0x6142
inicio: 0x6143

~~resto: 0x614E~~
~~comparo~~

Para las siguientes etiquetas, calcule el monto de una cada instrucción.

inicio: MOV R1, dato1
MOV R2, [dato2]
ADD R1, [R1]
resto: SUB [R2], [[cont]]
comparo: CMP R1, R3
JE fin
inc: ADD R3, 1
JMP comparo
fin: MOV [R2], FFFF

1 2 palabras, 4 posiciones
1 (0x6145) 2 palabras, 4 posiciones
1 (0x6147) 1 palabra, 2 posiciones
1 (0x6148) 2 palabras, 4 posiciones
1 (0x614A) 1 palabra, 2 posiciones
1 (0x614B) 1 palabra, 2 posiciones
1 (0x614D) 2 palabras, 4 posiciones
1 (0x614F) 2 palabras, 4 posiciones
1 (0x6150) (0x6151) = 50

Emulaciones:

resto: 0x6150	inc: 0x6158
comparo: 0x6151	fin: 0x6160
resto: 0x6148	inc: 0x614D
comparo: 0x614A	fin: 0x6151

b- Ver hoja de seguimiento adjunta.

Planilla de Seguimiento

Valores iniciales:

PC	SP
6143	FFFF

R0	R1	R2	R3	R4	R5	R6	R7
0	0	0	0	0	0	0	0

Z	C	V	N
0	0	0	0

Memoria:

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+A	+B	+C	+D	+E	+F
6140	1C1B	4532	4533	1B40	6140	1BFF	6141	28B1	3C9B	6142	6B63				
4530	F503	ABCD	21E2	6530	B000	1B40	95B1	6925	A639	ECBD	FAPD	ABCD	D1B8	A631	F17B

	PC	SP	[SP+1]	IR	Instrucción - 1 ^{er} palabra (en bits)	PC	2 ^{da} palabra	PC	3 ^{era} palabra	PC	Instrucción decodificada
1	6143	FFFF	?	1B40	0001 1000 0100 0000	6144	6140	6145			MOV R1, 6140
	Ejecución		$R_1 \leftarrow 6140$								Flag ¹ Z C V N
2	6145			1B8B	0001 1000 1000 1000	6146	6141	6147			MOV R2, [6141]
	Ejecución		$R_2 \leftarrow [6141] = 4532$								Flag ¹ Z C V N
3	6147			28B1	0010 1000 1011 0001	6148					ADD R2, [R1]
	Ejecución		$R_2 \leftarrow R_2 + [R_1] = 4532 + 1C1B = 614A$								Flag ¹ Z C V N
4	6148			3C9B	0011 1100 1001 1000	6149	6142	614A			SUB [R2], [6142]
	Ejecución		$[R_2] \leftarrow [R_2] - [6142] = 6B63 - 6550 = 0333$ ✓								Flag ¹ Z C V N
5	614A			0333	0000 0011 0011 0011	614B					Instrucción inválida
	Ejecución		Fin de la ejecución. ✓								Flag ¹ Z C V N
6											
	Ejecución										Flag ¹ Z C V N
7											
	Ejecución										Flag ¹ Z C V N
8											
	Ejecución										Flag ¹ Z C V N

¹ Sólo deben completarse luego de la ejecución de una instrucción que los altere.